# VISION-LANGUAGE MODELS ARE JOINT DENOISERS IN ROBOTIC ACTION GENERATION

**Anonymous authors**
Paper under double-blind review

## ABSTRACT

Diffusion-based vision-language-action (VLA) models cascade pretrained vision-language models (VLMs) with diffusion action experts, but typically restrict the VLM to providing static conditioning. This separation limits how much pretrained perception and reasoning can guide action generation, especially in out-of-distribution settings. We introduce JADE, a **J**oint **A**ction **DE**noiser that enables the VLM and action expert to jointly refine actions. Our key insight is that structured action latents allow the VLM to read the current noisy action together with visual observations and language instructions. At every denoising step, the VLM contextualizes the noisy action and guides the action expert through gated cross-attention, thereby its guidance evolves with the action being generated. We further supervise the VLM to recover clean action latents from noisy ones, which gives it a direct action-learning signal and couples perception with control. Additionally, joint denoising enables JADE to generate actions with fewer sampling steps, helping offset the additional cost of VLM participation. An adaptive step-size schedule further balances task performance and inference latency. On RoboDojo, JADE achieves an average success rate of 12.67%, the highest among the evaluated $\pi_{0.5}$-based methods, and is competitive with recent VLA and WAM foundation models pretrained at scale. It also improves performance on RoboTwin 2.0 and raises the average success rate across four real-world bimanual tasks by 20.8 percentage points compared with $\pi_{0.5}$.

## 1 INTRODUCTION

Early vision-language-action (VLA) models (Brohan et al., 2023; Kim et al., 2024) build on pretrained vision-language models (VLMs) and generate actions through next-token prediction. However, such autoregressive prediction suffers from slow inference and limited action precision. Along another line of research, diffusion policy (Chi et al., 2025) produces multimodal actions with higher speed and precision. Motivated by these complementary strengths, recent works combine the understanding capability of pretrained VLMs with the multimodal modeling capability of diffusion or flow action experts (Black et al., 2024; Intelligence et al., 2025; Bjorck et al., 2025). In these models, the VLM typically serves only as a static condition during both training and inference, while the action expert performs the entire denoising process. This design can be sufficient for in-domain tasks, where the model mainly learns relatively simple mappings from visual and language inputs to actions. However, it decouples perception from control. When facing out-of-distribution tasks, the action expert cannot effectively leverage the perception and reasoning capabilities of the VLM, which often leads to characteristic failure modes (Chen et al., 2026; Nasiriany et al., 2026). In addition, the VLM learns about actions only through gradients passed back from the action expert. The VLM should therefore dynamically participate in action generation rather than only provide a static condition.

As mentioned above, given the current visual state and a language instruction, pretrained VLMs can plan actions and be adapted into strong action generation models. Furthermore, prior work (Cen et al., 2025) shows that given the current clean action as a condition, the model can inversely predict the next visual state. In image generation, RepFusion (Pan et al., 2026) also lets the semantic condition change with the current noisy state during denoising. Here, we take these observations as a design principle for action generation. If a VLM can perceive clean action representations, can it also process their noisy counterparts during denoising?



Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

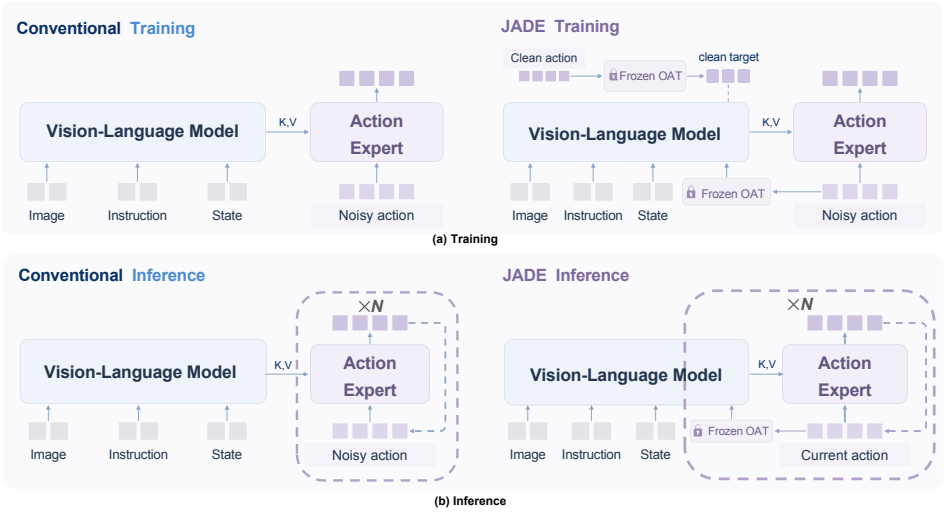



Figure 1: **Illustration of the key difference between our proposed JADE and conventional VLA models.** (a) During training, a frozen OAT encoder (Liu et al., 2026a) in JADE provides noisy action latents as VLM inputs and clean action latents as training targets. (b) During inference, JADE brings the VLM into the denoising loop, so its guidance to the action expert is updated with the current action at every step.

Answering this question requires noisy actions to be represented in a form that is compatible with the VLM and has meaningful semantic structure, which recent action tokenizers (Pertsch et al., 2025; Zhou et al., 2026; Dong et al., 2026; Liu et al., 2026a;b) make possible. Based on this idea, we propose JADE (**J**oint **A**ction **DE**noiser), a VLA model in which a pretrained VLM denoises actions jointly with the action expert, as illustrated in Figure 1. Our approach consists of three key designs. (1) In addition to visual and textual inputs, we feed compact latents of the current noisy action into the pretrained VLM. At every denoising step, the resulting VLM features condition the action expert, which denoises the action in its native action space. This design lets the VLM apply its strength in understanding structured representations to action generation. (2) To further unify perception and control, we supervise the VLM to predict clean action latents from noisy ones, which provides it with a direct action-learning signal from demonstrations. (3) Because each denoising step is better informed, JADE can use far fewer steps, which largely offsets the extra latency of joint denoising, and an adaptive step-size schedule further balances performance and efficiency.

The main contributions of this work can be summarized as follows:

- We propose JADE, a VLA framework in which the VLM reads the current noisy action at every denoising step and guides the action expert. This lets the VLM provide dynamic denoising priors beyond static vision-language conditioning.
- We find that joint denoising makes action generation easier, so JADE can use far fewer denoising steps than conventional flow-based VLAs. With an adaptive step-size schedule, JADE further balances performance and efficiency.
- We show that JADE consistently improves over its $\pi_{0.5}$ (Intelligence et al., 2025) base on RoboTwin 2.0 (Chen et al., 2025b) , RoboDojo (Chen et al., 2026), and four real-world bimanual tasks, and ablations verify the effectiveness of its key designs.

## 2 Related Work

### 2.1 Vision-Language-Action Models

Early VLAs discretize continuous controls and predict action tokens within the language-model sequence (Brohan et al., 2022; 2023; Kim et al., 2024). More recent VLAs use diffusion or flow action generators, where features from a pretrained vision-language backbone condition a continuous action expert (Black et al., 2024; Intelligence et al., 2025; Bjorck et al., 2025). We follow this

paradigm but enable the VLM to dynamically participate in the iterative refinement of the evolving action.

**Action Tokenizers.** Action tokenizers connect continuous control with sequence modeling. Early approaches quantize each action dimension independently (Brohan et al., 2023; Kim et al., 2024), and later methods exploit temporal structure through frequency-domain compression, learned discrete codes, or structured latent representations (Pertsch et al., 2025; Wang et al., 2025; Dong et al., 2026; Liu et al., 2026a). JADE builds on OAT (Liu et al., 2026a) and uses its compact latents to let the VLM read the current noisy action.

### 2.2 Integration of Language Models and Denoisers

Multimodal generative models increasingly combine pretrained semantic models with diffusion or flow generators, through learned queries (Pan et al., 2025; Chen et al., 2025a), through shared attention, Mixture-of-Transformers, and unified architectures (Shi et al., 2026; Deng et al., 2025; Zhou et al., 2025; Ma et al., 2025), or through conditions that change with the current noisy state (Pan et al., 2026). Robot learning shows a similar trend, where VLMs are coupled with continuous action generators through attention, shared representations, or intermediate action abstractions (Black et al., 2024; Bjorck et al., 2025; Bi et al., 2026; Agarwal et al., 2026; Li et al., 2026c). However, in many modular diffusion and flow VLAs, the vision-language features used for action generation do not depend on the evolving noisy action. JADE instead contextualizes the current noisy action with vision and language in the VLM and feeds the resulting features to the action expert at every denoising step.

## 3 Method

### 3.1 Preliminaries

**Flow-based VLA Policies.** Given a multimodal observation $o_t$ consisting of images, a language instruction, and robot state, a flow-based VLA policy couples a pretrained VLM of hidden width $W_v$ with a continuous action expert to predict a future action chunk $\mathbf{A} = [\mathbf{a}_t, \dots, \mathbf{a}_{t+H-1}] \in \mathbb{R}^{H\times D}$ (Black et al., 2024; Intelligence et al., 2025). The width $D$ includes padded dimensions, and the superscript $\mathrm{phys}$ keeps the $D_p$ valid physical dimensions of any action-shaped tensor, such as $\mathbf{A}^{\mathrm{phys}} \in \mathbb{R}^{H\times D_p}$. Training uses conditional flow matching, interpolating clean actions with Gaussian noise as

$$\mathbf{A}_\tau = (1-\tau)\mathbf{A} + \tau\boldsymbol{\epsilon}, \qquad \boldsymbol{\epsilon} \sim \mathcal{N}(0, I). \tag{1}$$

The action expert predicts a velocity $v_\Theta(\mathbf{A}_\tau, \tau, o_t)$ that regresses $\boldsymbol{\epsilon} - \mathbf{A}$, and at inference this velocity is integrated from Gaussian noise at $\tau = 1$ toward clean actions at $\tau = 0$.

**Ordered Action Tokenization.** OAT (Liu et al., 2026a) uses a transformer encoder with $M$ learnable registers to aggregate information across an action chunk into compact continuous latents $E_{\mathrm{OAT}}(\mathbf{A}^{\mathrm{phys}}) = [\mathbf{z}_1, \dots, \mathbf{z}_M] \in \mathbb{R}^{M\times d_z}$. Through nested dropout, its training encourages an ordered representation, with earlier registers capturing coarse action structure and later registers encoding finer details. The original OAT framework discretizes these latents using finite scalar quantization (FSQ), while JADE uses their continuous values before rounding as the action interface to the VLM. We pretrain OAT through action reconstruction in the same normalized action space as the policy and keep it frozen, and we write $\mathbf{Z}_\tau = E_{\mathrm{OAT}}(\mathbf{A}_\tau^{\mathrm{phys}})$ for the latents of a noisy action and $\mathbf{Z}_0 = E_{\mathrm{OAT}}(\mathbf{A}^{\mathrm{phys}})$ for those of the clean action. We choose OAT because its few compact registers keep the per-step VLM cost low and it performs best among the action representations we tested. Other representations also improve over $\pi_{0.5}$, as Section 4.3 shows, so the benefit of joint denoising does not hinge on this choice. Appendices A.1 and A.2 give more details.

### 3.2 Noisy Action Representations for VLM

As shown in Figure 2(b), JADE uses the VLM to transform the current noisy action into observation-conditioned register features at each denoising step. Given a noisy action $\mathbf{A}_\tau$, the frozen OAT

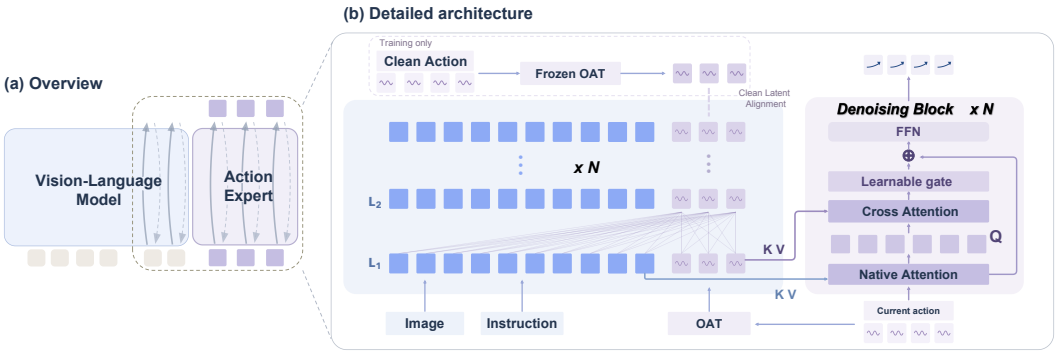



Figure 2: **Architecture of JADE.** A frozen OAT encoder maps the current noisy action to compact latents, which the VLM processes together with the observation prefix. The resulting register features guide each denoising block of the action expert through gated cross-attention, and during training the final features are aligned with the clean action latents.

encoder produces continuous latents $\mathbf{Z}_\tau$, and a trainable input adapter $P_\eta$ maps them to the VLM input $\mathbf{X}_\tau = P_\eta(\mathbf{Z}_\tau, \tau) \in \mathbb{R}^{M \times W_v}$.

The same VLM that encodes the observation prefix then processes these registers, and we denote the layer-wise keys and values of the prefix by $\mathcal{C}(o_t)$. At each layer, the registers attend to valid prefix tokens and bidirectionally to one another, while the prefix does not attend to them, so $\mathcal{C}(o_t)$ is computed once per observation and reused across denoising steps. Each register hidden state further passes through a bottleneck residual adapter shared across layers, which preserves the compact register sequence while adapting its features. We write the resulting register features at each of the $L$ layers and at the final output as

$$\{\mathbf{R}_\tau^{(\ell)}\}_{\ell=1}^{L},\ \mathbf{R}_\tau^{\text{out}} = F_\phi\big(\mathbf{X}_\tau;\, \mathcal{C}(o_t)\big), \tag{2}$$

where $F_\phi$ denotes the VLM computation together with the shared adapter, which Appendix B.2 describes in detail.

To give the VLM a direct action-learning signal, a linear head $G_\psi$ predicts the clean OAT latents $\mathbf{Z}_0$ from the final register features with

$$\mathcal{L}_{\text{align}} = \text{SmoothL1}\left(G_\psi(\mathbf{R}_\tau^{\text{out}}),\, \mathbf{Z}_0\right), \tag{3}$$

averaged over registers and latent dimensions. The VLM thus learns to recover clean action latents from the observation and a noisy action. Because the adapter is shared across layers, the structure it learns under this supervision also shapes the layer-wise features $\{\mathbf{R}_\tau^{(\ell)}\}_{\ell=1}^{L}$, which guide the action expert as described in Section 3.3.

### 3.3 Joint Denoising with the VLM

We incorporate the action-state-dependent VLM features from Section 3.2 into the action expert, allowing both pathways to contribute to velocity prediction. In each native expert layer, action queries attend to the valid observation prefix and all action positions, with attention weights normalized over the combined key set. The observation prefix does not attend to actions.

As shown in Figure 2(b), JADE inserts a separate cross-attention module after the native attention residual update and before the feed-forward network in each expert layer. This module reads the VLM register features associated with the current action. Let $\mathbf{H}_\tau^{(\ell)}$ denote the action hidden states after the native attention update and $\mathbf{R}_\tau^{(\ell)}$ the corresponding VLM features. The fusion is computed as

$$\begin{aligned} \mathbf{U}_\tau^{(\ell)} &= \text{CrossAttn}_\ell\left(\text{RMSNorm}(\mathbf{H}_\tau^{(\ell)}),\, T_\ell(\mathbf{R}_\tau^{(\ell)})\right), \\ \widetilde{\mathbf{H}}_\tau^{(\ell)} &= \mathbf{H}_\tau^{(\ell)} + \alpha_\tau^{(\ell)} \mathbf{U}_\tau^{(\ell)}. \end{aligned} \tag{4}$$

Here, $T_\ell$ maps the VLM features into the representation space required by the layer's fusion module. Cross-attention uses the action hidden states as queries and the mapped VLM register features as

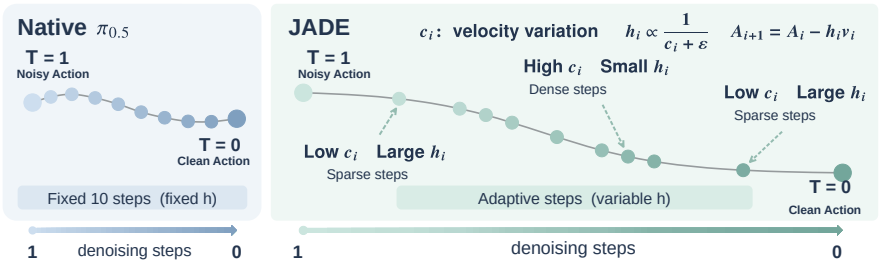



Figure 3: **Adaptive inference for joint denoising.** The left panel shows uniform Euler steps, which native $\pi_{0.5}$ and default JADE both use. The right panel shows the adaptive schedule, where a small $c_i$ gives a large step and a large $c_i$ gives a small step. Trajectories are schematic.

keys and values, allowing each action position to read all registers. Its output is projected back to the expert's hidden width and added through a gated residual connection. This module normalizes attention independently over the register keys, preserving the native attention's key–value set and normalization domain. The fused states then enter the feed-forward network in the same layer.

To regulate the VLM-derived residual update, we combine a layer-specific parameter with a state-dependent dynamic gate:

$$\alpha_\tau^{(\ell)} = s_g \tanh\Big[b_\ell + g_\omega\Big(\mathrm{Avg}(\mathbf{H}_\tau^{(\ell)}), \mathrm{Avg}(\mathbf{U}_\tau^{(\ell)}), \mathbf{e}_g(\tau)\Big)\Big]. \tag{5}$$

Here, Avg denotes averaging over action positions, $b_\ell$ is a layer-specific parameter, $g_\omega$ is a gating network shared across layers, $\mathbf{e}_g(\tau)$ is a time embedding of the noise level, and $s_g$ is a scaling factor. The gate produces one scalar residual coefficient per sample and layer. We initialize $b_\ell$ and the output layer of $g_\omega$ to zero, making the added fusion residuals initially zero and preserving the native expert computation at initialization.

After all expert layers, the model outputs the action-space velocity $v_\Theta(\mathbf{A}_\tau, \tau, o_t)$. We retain the native flow-matching objective and jointly optimize it with the clean-action representation supervision defined in Section 3.2:

$$\mathcal{L}_{\text{flow}} = \mathrm{MSE}\left(v_\Theta(\mathbf{A}_\tau, \tau, o_t),\, \boldsymbol{\epsilon} - \mathbf{A}\right), \qquad \mathcal{L} = \mathcal{L}_{\text{flow}} + \lambda_{\text{align}}\mathcal{L}_{\text{align}}. \tag{6}$$

Both objectives train the shared VLM and the input and residual adapters, with flow-matching gradients reaching the VLM through the fusion modules. The OAT encoder remains frozen.

At inference, we cache the observation-prefix keys and values $\mathcal{C}(o_t)$ once per policy query. The native Euler sampler integrates from Gaussian noise at $\tau = 1$ to $\tau = 0$ in the native action space. Each velocity evaluation recomputes the OAT latents and layer-wise VLM features from the current noisy action and uses the fusion-augmented expert to predict the velocity.

### 3.4 Adaptive Inference for Joint Denoising

Joint denoising evaluates the VLM at every step, so its total cost grows with the number of velocity evaluations despite prefix caching and compact action registers. Unlike conventional flow-based VLAs, JADE conditions the VLM on the current noisy action at each step. This action-dependent guidance enables generation with fewer velocity evaluations, offsetting the added per-step cost. The full joint model is evaluated at every step.

Building on fewer steps, we further let the sampler choose the step size adaptively. It takes larger steps when the predicted corrections are stable and smaller steps when they fluctuate, as shown in Figure 3. At the $i$-th velocity evaluation, we define

$$\mathbf{v}_i = v_\Theta(\mathbf{A}_{\tau_i}, \tau_i, o_t), \qquad \mathbf{q}_i = -\tau_i\, \mathbf{v}_i^{\text{phys}}. \tag{7}$$

Since $\mathbf{A}_{\tau_i} - \tau_i \mathbf{v}_i$ is the clean action implied by the current velocity, the time-weighted correction $\mathbf{q}_i$ is the displacement from the current action toward this clean estimate. We track the first and second moments of $\mathbf{q}_i$ with exponential moving averages. Both moments are initialized as $\mathbf{m}_{-1} = \mathbf{s}_{-1} = \mathbf{0}$ at the beginning of each policy query and are updated as

$$\begin{gathered}\mathbf{m}_i = \beta \mathbf{m}_{i-1} + (1-\beta)\mathbf{q}_i, \qquad \mathbf{s}_i = \beta \mathbf{s}_{i-1} + (1-\beta)\mathbf{q}_i^{\odot 2}, \\ c_i = \left\| [\mathbf{s}_i - \mathbf{m}_i^{\odot 2}]_+ \right\|_F^{1/2}.\end{gathered} \tag{8}$$

where $\beta$ is the decay coefficient, $\odot 2$ denotes elementwise squaring, and $[\cdot]_+$ clips negative values to zero. Therefore, $c_i$ measures how much the corrections have fluctuated over recent steps.

The first step uses a fixed step size $h_0$ and updates the statistics. For $i \geq 1$, the step size is set from $c_i$, and the action and flow time are updated by Euler integration as

$$h_i = \min\left\{\tau_i,\ \mathrm{clip}\left(\frac{\lambda_{\mathrm{step}}}{c_i + \varepsilon_c}, h_{\min}, h_{\max}\right)\right\}, \qquad \mathbf{A}_{\tau_{i+1}} = \mathbf{A}_{\tau_i} - h_i \mathbf{v}_i, \qquad \tau_{i+1} = \tau_i - h_i, \tag{9}$$

Here, $\lambda_{\mathrm{step}}$ controls the step-size scale, $\varepsilon_c > 0$ stabilizes the denominator, and $h_{\min}$ and $h_{\max}$ are clipping bounds. Smaller $c_i$ permits larger steps, capped by the remaining time $\tau_i$. Statistics use physical action dimensions, while Euler updates use the full action tensor.

The step-size statistics require no additional model evaluations, and both model weights and the robot execution horizon remain unchanged. We use $c_i$ only to guide sampling, not to estimate task difficulty or success probability. Section 4.4 evaluates the performance–latency trade-off across step counts and compares fixed-step and adaptive sampling using the same checkpoint. Appendix C.2 provides mathematical analysis and proofs of basic properties of the adaptive sampling rule.

# 4 Experiments

## 4.1 Experimental Setup

**Benchmarks and metrics.** We evaluate closed-loop manipulation on RoboTwin 2.0 (Chen et al., 2025b) and RoboDojo (Chen et al., 2026) following their official settings. On RoboTwin 2.0, a single multi-task policy is trained on clean demonstrations of 50 bimanual tasks and evaluated under Clean and Random scene conditions. The Random scenes are not seen during training. On RoboDojo, the policy is trained on 3,500 demonstrations and evaluated on 42 tasks in five capability categories. We report success rates on both benchmarks and the official score on RoboDojo.

**Implementation details.** We build on $\pi_{0.5}$ with three RGB views, language instructions, robot states, and a 50-step action chunk. For each benchmark, we pretrain and freeze an OAT encoder with eight four-dimensional registers, and we train all other components jointly with $\lambda_{\mathrm{align}} = 0.1$. Unless otherwise specified, inference uses ten Euler steps and executes the first 25 actions of each chunk before replanning. We also report a two-step variant and JADE Balanced, which uses the adaptive schedule in Section 3.4. Appendix D gives the full evaluation protocol and training details.

## 4.2 Main Results

On RoboTwin 2.0 (Table 1), JADE achieves the highest Random and average success rates among the listed methods, at 54.4% and 66.9%. Many methods with high success rates under Clean degrade sharply under Random. In contrast, JADE improves the Random success rate of $\pi_{0.5}$ by 8.4 percentage points. These results suggest that the VLM's dynamic participation in action generation improves robustness to visual randomization by updating its conditioning features as the action candidate evolves.

On RoboDojo (Table 2), JADE ranks highest on both overall metrics and improves over $\pi_{0.5}$ in all five capability categories. The largest gains appear in Precision and Long-Horizon, and the Precision success rate nearly triples. At every step, the VLM reads the current action candidate together with the instruction, which helps correct fine deviations and keep the actions consistent with the current task stage. This suggests that the planning ability of the VLM improves the precision and coherence of action generation and thereby strengthens long-horizon execution. Together, these results corroborate that action-conditioned VLM representations help translate scene and task information into effective action refinement. Appendix D.2 reports the per-task results on both benchmarks.

## 4.3 Ablation Study

**Action-state conditioning.** Table 3(a) replaces the OAT latents of the current noisy action with the same number of learnable queries. Action latents consistently outperform the queries on both benchmarks, by up to 7.9 percentage points on RoboTwin. Action latents are recomputed from the

Table 1: Results on the RoboTwin 2.0 benchmark. Success rates (%) are averaged over 50 tasks under Clean and Random settings; Avg. is their mean. JADE uses 10 denoising steps. Best values in each column are bold.

| Method | Clean | Random | Avg. |
|---|---|---|---|
| ACT (Zhao et al., 2023) | 29.7 | 1.7 | 15.7 |
| RDT (Liu et al., 2025) | 34.5 | 13.7 | 24.1 |
| $\pi_0$ (Black et al., 2024) | 46.4 | 16.3 | 31.4 |
| GR00T-N1.7 (Bjorck et al., 2025) | 43.6 | 20.7 | 32.2 |
| StarVLA (Community, 2026) | 58.1 | 10.6 | 34.4 |
| FastWAM (Yuan et al., 2026) | 70.2 | 1.2 | 35.7 |
| X-VLA (Zheng et al., 2026) | 68.0 | 20.9 | 44.5 |
| Lingbot-VA (Li et al., 2026b) | 80.7 | 34.6 | 57.7 |
| 4D-WAM (Yang et al., 2026) | **81.5** | 41.8 | 61.7 |
| $\pi_{0.5}$ (Intelligence et al., 2025) | 68.5 | 46.0 | 57.3 |
| **JADE (ours)** | 79.3 | **54.4** | **66.9** |

Table 2: Results on the RoboDojo benchmark. Each entry reports *Score / Success Rate (SR, %)*. JADE uses 10 denoising steps.

| Method | Gen. | Prec. | Long. | Mem. | Open | Avg. |
|---|---|---|---|---|---|---|
| OpenVLA-OFT (Kim et al., 2025) | 0.04 / 0.00 | 0.20 / 0.00 | 0.70 / 0.00 | 0.00 / 0.00 | 0.08 / 0.08 | 0.21 / 0.02 |
| RDT-1B (Liu et al., 2025) | 0.56 / 0.33 | 0.38 / 0.00 | 1.13 / 0.00 | 0.49 / 0.33 | 0.00 / 0.00 | 0.51 / 0.13 |
| GR00T-N1.7 (Bjorck et al., 2025) | 2.16 / 1.22 | 2.54 / 0.67 | 8.30 / 3.58 | 1.06 / 0.89 | 0.18 / 0.17 | 2.85 / 1.31 |
| $\pi_0$ (Black et al., 2024) | 3.94 / 2.56 | 3.56 / 0.75 | 6.19 / 2.00 | 3.47 / 2.11 | 0.25 / 0.25 | 3.48 / 1.53 |
| Fast-WAM (Yuan et al., 2026) | 2.34 / 1.11 | 1.96 / 0.00 | 9.14 / 5.17 | 3.55 / 3.44 | 0.42 / 0.42 | 3.48 / 2.03 |
| LingBot-VLA (Wu et al., 2026) | 6.71 / 4.28 | 5.33 / 1.83 | 10.89 / 5.25 | 3.82 / 2.78 | 0.72 / 0.67 | 5.50 / 2.96 |
| StarVLA-$\alpha$ (Ye et al., 2026) | 3.93 / 2.33 | 9.90 / 4.33 | 14.15 / 6.50 | 3.34 / 2.44 | 0.68 / 0.58 | 6.40 / 3.24 |
| X-VLA (Zheng et al., 2026) | 10.48 / 6.78 | 18.32 / 12.00 | 16.53 / 9.75 | 4.76 / 3.56 | 0.55 / 0.50 | 10.13 / 6.52 |
| Spatial Forcing (Li et al., 2026a) | 14.12 / 9.33 | 17.33 / 10.58 | 23.26 / 14.58 | 5.43 / 4.11 | 1.78 / 1.58 | 12.38 / 8.04 |
| $\pi_{0.5}$ (Intelligence et al., 2025) *(base)* | 13.37 / 8.17 | 12.40 / 5.50 | 23.54 / 14.67 | 5.78 / 4.56 | 1.98 / 1.67 | 11.41 / 6.91 |
| **JADE (ours)** | **20.59 / 14.06** | **23.41 / 16.33** | **36.55 / 24.00** | **7.87 / 6.56** | **3.05 / 2.42** | **18.29 / 12.67** |

current noisy action at every step, so they are tightly coupled with the generation process and let the VLM respond to the specific candidate it is refining.

**Auxiliary alignment.** Table 3(a) also measures the effect of alignment by training each variant with and without $\mathcal{L}_{\text{align}}$. Alignment further improves JADE on both benchmarks. By training the VLM to recover clean action latents from noisy ones, it gives the VLM a direct action-learning signal and makes the features passed to the action expert more informative about the target action.

**Action representation.** Table 3(b) evaluates how the action representation fed to the VLM affects JADE. All evaluated representations improve the average RoboTwin success rate over $\pi_{0.5}$, which shows that the benefit of joint denoising does not depend on a specific representation interface. OAT achieves the best performance with only eight compact registers, outperforming G0.5 2-RVQ (Liu et al., 2026b) with 166 tokens, and its advantage is most pronounced under Random. This indicates that the effectiveness of an action representation depends not only on its capacity but also on whether its structure matches the semantic modeling ability of the VLM. Compact action representations with semantic organization are thus better suited as the interface through which the VLM reads the noisy action, and they provide more effective guidance to the action expert. FAST (Pertsch et al., 2025) represents actions with frequency coefficients and lacks an explicit semantic organization, so it yields the lowest average success rate.

### 4.4 Computational Cost and Sampling Efficiency

JADE changes the cost structure of VLA inference. Each denoising step reruns the VLM on the current action, so a single step costs more but also carries more information for correcting the action. Prefix caching and compact registers keep this per-step cost low, and a full training update takes about 23% longer in a matched microbenchmark.

To test whether these more informative steps allow far fewer of them, we fix the checkpoint and the prediction and execution horizons and vary only the number of function evaluations, or NFE, used by Euler sampling on fixed task subsets. As shown in the left panel of Figure 4, native $\pi_{0.5}$ degrades

Table 3: **Ablation studies.** We examine (a) action conditioning and auxiliary alignment supervision, (b) action representations, and (c) step-wise action-conditioned semantic updates. RoboTwin reports success rates (%); RoboDojo in (a) reports SR and Score averaged over its five evaluation dimensions, while (c) reports Score and full-task success rate on RoboDojo.

**(a) Conditioning and Alignment**

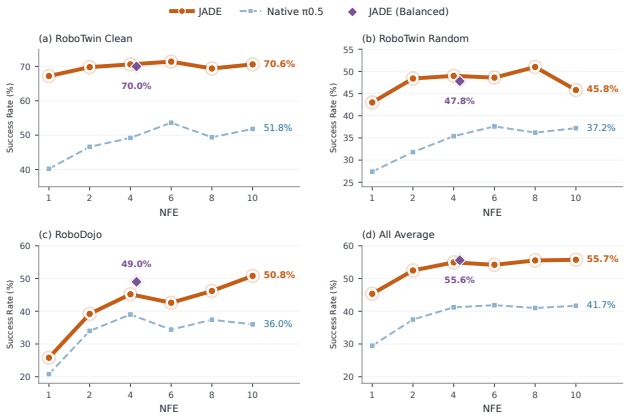
| | | RoboTwin | | | RoboDojo | |
|---|---|---|---|---|---|---|
| Conditioning | $\lambda_{\text{align}}$ | Clean | Random | Avg. | SR | Score |
| Queries | 0 | 73.1 | 49.5 | 61.3 | 7.68 | 11.33 |
| Queries | 0.1 | 72.6 | 45.3 | 59.0 | 9.66 | 15.36 |
| Action latents | 0 | 79.1 | 52.9 | 66.0 | 11.15 | 16.59 |
| Action latents | 0.1 | 79.3 | 54.4 | 66.9 | 12.67 | 18.29 |

**(b) Action Representation**

| | RoboTwin | | |
|---|---|---|---|
| Representation | Clean | Random | Avg. |
| Bin256 | 73.5 | 46.8 | 60.2 |
| G0.5 2-RVQ | 77.3 | 46.3 | 61.8 |
| FAST | 74.8 | 44.5 | 59.7 |
| OAT (matched) | 79.3 | 54.4 | 66.9 |

**(c) Semantic Updates**

| | RoboDojo | |
|---|---|---|
| Variant | Score | SR |
| Dynamic | **23.2** | **15.0** |
| Static | 19.3 | 11.9 |
| Lagged | 20.8 | 14.3 |
| Fixed | 19.7 | 12.1 |
| RF-Off | 18.1 | 11.1 |




*RoboTwin 2.0, SR (%)*

| Config. | NFE | Clean | Random | Avg. |
|---|---|---|---|---|
| 10-step | 10 | 79.3 | 54.4 | 66.9 |
| 2-step | 2 | 80.1 | 51.8 | 66.0 |
| Balanced | 4.995 | 77.9 | 52.4 | 65.2 |

*RoboDojo*

| Config. | NFE | Score | SR |
|---|---|---|---|
| 10-step | 10 | 18.29 | 12.67 |
| 2-step | 2 | 14.52 | 9.67 |
| Balanced | 4.985 | 18.06 | 12.48 |

Figure 4: **Sampling efficiency of JADE.** The left panel shows the success rates of JADE and native $\pi_{0.5}$ across different numbers of function evaluations (NFE) on fixed task subsets, and purple diamonds mark JADE Balanced. The right table reports three sampling configurations on the full benchmarks, with NFE averaged per action chunk.

markedly as the NFE decreases, whereas JADE with only two steps already matches or exceeds ten-step native $\pi_{0.5}$. The right table confirms this pattern on the full benchmarks, where two-step JADE reaches 66.0% on RoboTwin and 9.67% on RoboDojo, compared with 57.3% and 6.91% for ten-step $\pi_{0.5}$. With the VLM taking part in every step, a few well-informed steps can replace many weakly informed ones.

To balance performance and sampling cost, we also evaluate the adaptive Balanced schedule. On the full RoboDojo benchmark, it retains 98.5% of JADE's ten-step success rate with about five evaluations on average.

The higher per-step cost is thus largely offset by the smaller number of steps. On the model side, JADE takes 203 ms per policy query with ten steps, 132 ms with Balanced, and 78 ms with two steps, compared with about 74 ms for ten-step native $\pi_{0.5}$. Two-step JADE is therefore only slightly slower than native $\pi_{0.5}$ while performing clearly better, and JADE offers a tunable balance between performance and efficiency rather than a fixed slowdown. Its remaining overhead buys stronger perception and reasoning during action generation, similar in spirit to VLAs that reason before acting, without decoding reasoning text. Appendix E reports the operation counts, task subsets, and timing conditions.

### 4.5 In-depth Analysis of Joint Denoising

To further examine the benefits of dynamic VLM participation in denoising, we compare five inference-time variants of the same trained model. *Dynamic*, the default, recomputes the VLM features from the current action at every step, *Lagged* reuses those from the previous step, *Fixed* recomputes them from the initial noise with the current flow time, *Static* computes them once from the initial noise, and *RF-Off* removes the injection. All variants are evaluated on the same 100 sce-

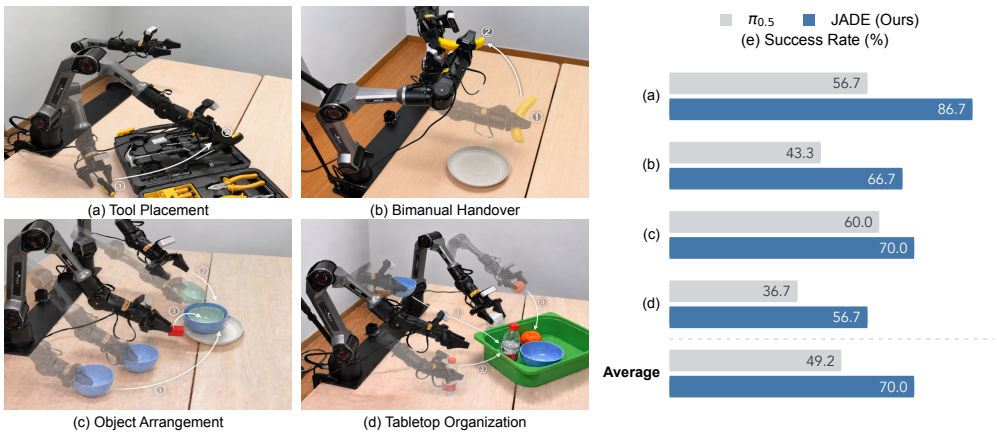



Figure 5: Real-robot task illustrations and success-rate comparisons. Panels (a)–(d) show the four tasks, and panel (e) compares JADE with $\pi_{0.5}$.

narios for each of 10 RoboDojo tasks, two per capability category, which differ from the subsets in Section 4.4. Appendix F gives the full details.

Table 3(c) shows that performance rises steadily as the VLM features follow the current action more closely, from RF-Off through Static, Fixed, and Lagged to Dynamic, which reaches a Score of 23.2 and a full-task success rate of 15.0%. Static improves only slightly over RF-Off, and Fixed adds little on top of Static, so neither a fixed VLM context nor time conditioning accounts for most of the gain. What matters most is whether the features track the action itself, as moving from Fixed to Dynamic contributes 2.9 of the 3.9 percentage points that separate RF-Off from Dynamic. Lagged recovers much of this gain by reading the action from the previous step, yet it remains behind Dynamic, as guidance computed from a stale action no longer matches the candidate being refined. The VLM thus acts less like a fixed source of context and more like an evaluator that reexamines the current action at each step, and its guidance is most useful when it describes the exact candidate that the action expert is about to refine.

### 4.6 Real-world Experiments

We further evaluate JADE on a PiPER dual-arm platform across four tasks, namely Tool Placement, Bimanual Handover, Object Arrangement, and Tabletop Organization. For each task, we train $\pi_{0.5}$ and JADE on the same 100 teleoperated demonstrations and evaluate each policy over 30 trials, with details in Appendix G. As shown in Figure 5, JADE outperforms $\pi_{0.5}$ on every task and raises the overall success rate from 49.2% to 70.0%. The gains are largest on Tool Placement, Bimanual Handover, and Tabletop Organization, which require precise placement, coordination between the two arms, and execution over several stages. These tasks depend on correcting the action as it unfolds, which JADE supports by letting the VLM reexamine the full bimanual action candidate against the scene at every denoising step.

## 5 Conclusion

We presented JADE, a philosophy that enables pretrained VLMs to actively participate in action denoising. By processing structured noisy action representations and learning to predict clean action latents under direct supervision, the VLM provides dynamic guidance to the action expert, tightly coupling perception and control throughout action generation. Experiments on RoboTwin 2.0, RoboDojo, and real-world bimanual tasks demonstrate consistent improvements over the baseline. Further analyses demonstrate the advantages of involving the VLM in joint denoising. These results demonstrate the potential of extending VLMs beyond static conditioning and show that their active involvement in action generation offers a critical solution for improving robotic reasoning and generalization.

### AI USE STATEMENT

Generative AI tools were used to assist with manuscript organization and English drafting. They were not used to generate experimental data, execute the reported evaluations, or replace the authors' development of the proposed method. All AI-assisted technical descriptions, citations, numerical claims, and interpretations were verified by the authors against the source code, experiment records, and original research materials before submission. The authors take responsibility for the final content of this work.

### ETHICS STATEMENT

This work studies vision-language-action models for robotic manipulation in simulation and real-world settings. The experiments do not involve human participants or personal data. Nevertheless, robotic systems deployed in physical environments may introduce safety risks if instructions are misinterpreted or control failures occur under distribution shifts. Real-world deployment and evaluation should therefore incorporate appropriate workspace constraints, emergency-stop procedures, and human supervision according to the robotic platform.

### REPRODUCIBILITY STATEMENT

The manuscript describes the proposed model formulation, training procedure, and evaluation protocols. Appendix B provides detailed implementation configurations, Appendix D describes benchmark settings and evaluation details, and Appendix E reports computational cost and timing analysis. Additional task-level results and real-world experimental details are provided in Appendices D and G, respectively. These materials facilitate verification and reproduction of the reported results.

## REFERENCES

Niket Agarwal, Arslan Ali, Jon Allen, Martin Antolini, Adeline Aubame, Alisson Azzolini, Junjie Bai, Maciej Bala, Yogesh Balaji, Josh Bapst, et al. Cosmos 3: Omnimodal world models for physical ai. *arXiv preprint arXiv:2606.02800*, 2026.

Hongzhe Bi, Hengkai Tan, Shenghao Xie, Zeyuan Wang, Shuhe Huang, Haitian Liu, Ruowen Zhao, Yao Feng, Chendong Xiang, Yinze Rong, et al. Motus: A unified latent action world model. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 35101–35113, 2026.

Johan Bjorck, Fernando Castañeda, Nikita Cherniadev, Xingye Da, Runyu Ding, Linxi Fan, Yu Fang, Dieter Fox, Fengyuan Hu, Spencer Huang, et al. Gr00t n1: An open foundation model for generalist humanoid robots. *arXiv preprint arXiv:2503.14734*, 2025.

Kevin Black, Noah Brown, Danny Driess, Adnan Esmail, Michael Equi, Chelsea Finn, Niccolo Fusai, Lachy Groom, Karol Hausman, Brian Ichter, et al. $\pi_0$: A vision-language-action flow model for general robot control. *arXiv preprint arXiv:2410.24164*, 2024.

Anthony Brohan, Noah Brown, Justice Carbajal, Yevgen Chebotar, Joseph Dabis, Chelsea Finn, Keerthana Gopalakrishnan, Karol Hausman, Alex Herzog, Jasmine Hsu, et al. Rt-1: Robotics transformer for real-world control at scale. *arXiv preprint arXiv:2212.06817*, 2022.

Anthony Brohan, Noah Brown, Justice Carbajal, Yevgen Chebotar, Xi Chen, Krzysztof Choromanski, Tianli Ding, Danny Driess, Avinava Dubey, Chelsea Finn, et al. Rt-2: Vision-language-action models transfer web knowledge to robotic control. *arXiv preprint arXiv:2307.15818*, 2023.

Jun Cen, Chaohui Yu, Hangjie Yuan, Yuming Jiang, Siteng Huang, Jiayan Guo, Xin Li, Yibing Song, Hao Luo, Fan Wang, et al. Worldvla: Towards autoregressive action world model. *arXiv preprint arXiv:2506.21539*, 2025.

Jiuhai Chen, Zhiyang Xu, Xichen Pan, Yushi Hu, Can Qin, Tom Goldstein, Lifu Huang, Tianyi Zhou, Saining Xie, Silvio Savarese, et al. Blip3-o: A family of fully open unified multimodal models-architecture, training and dataset. *arXiv preprint arXiv:2505.09568*, 2025a.

Tianxing Chen, Zanxin Chen, Baijun Chen, Zijian Cai, Yibin Liu, Zixuan Li, Qiwei Liang, Xianliang Lin, Yiheng Ge, Zhenyu Gu, et al. Robotwin 2.0: A scalable data generator and benchmark with strong domain randomization for robust bimanual robotic manipulation. *arXiv preprint arXiv:2506.18088*, 2025b.

Tianxing Chen, Yue Chen, Zixuan Li, Junyuan Tang, Kailun Su, Haoran Lu, Weijie Wan, Baijun Chen, Songling Liu, Haowen Yan, et al. Robodojo: A unified sim-and-real benchmark for comprehensive evaluation of generalist robot manipulation policies. *arXiv preprint arXiv:2607.04434*, 2026.

Cheng Chi, Zhenjia Xu, Siyuan Feng, Eric Cousineau, Yilun Du, Benjamin Burchfiel, Russ Tedrake, and Shuran Song. Diffusion policy: Visuomotor policy learning via action diffusion. *The International Journal of Robotics Research*, 44(10-11):1684–1704, 2025.

StarVLA Community. Starvla: A lego-like codebase for vision-language-action model developing. *arXiv preprint arXiv:2604.05014*, 2026.

Chaorui Deng, Deyao Zhu, Kunchang Li, Chenhui Gou, Feng Li, Zeyu Wang, Shu Zhong, Weihao Yu, Xiaonan Nie, Ziang Song, et al. Emerging properties in unified multimodal pretraining. *arXiv preprint arXiv:2505.14683*, 2025.

Zibin Dong, Yicheng Liu, Shiduo Zhang, Baijun Ye, Yifu Yuan, Fei Ni, Jingjing Gong, Xipeng Qiu, Hang Zhao, Yinchuan Li, et al. Actioncodec: What makes for good action tokenizers. *arXiv preprint arXiv:2602.15397*, 2026.

Physical Intelligence, Kevin Black, Noah Brown, James Darpinian, Karan Dhabalia, Danny Driess, Adnan Esmail, Michael Equi, Chelsea Finn, Niccolo Fusai, et al. $\pi_{0.5}$: a vision-language-action model with open-world generalization. *arXiv preprint arXiv:2504.16054*, 2025.

Moo Jin Kim, Karl Pertsch, Siddharth Karamcheti, Ted Xiao, Ashwin Balakrishna, Suraj Nair, Rafael Rafailov, Ethan Foster, Grace Lam, Pannag Sanketi, et al. Openvla: An open-source vision-language-action model. *arXiv preprint arXiv:2406.09246*, 2024.

Moo Jin Kim, Chelsea Finn, and Percy Liang. Fine-tuning vision-language-action models: Optimizing speed and success. *arXiv preprint arXiv:2502.19645*, 2025.

Fuhao Li, Wenxuan Song, Han Zhao, Jingbo Wang, Pengxiang Ding, Donglin Wang, Long Zeng, and Haoang Li. Spatial forcing: Implicit spatial representation alignment for vision-language-action model. In *International Conference on Learning Representations*, volume 2026, pp. 132324–132345, 2026a.

Lin Li, Qihang Zhang, Yiming Luo, Shuai Yang, Ruilin Wang, Fei Han, Mingrui Yu, Zelin Gao, Nan Xue, Xing Zhu, et al. Causal world modeling for robot control. *arXiv preprint arXiv:2601.21998*, 2026b.

Qiwei Li, Xicheng Gong, Xinghang Li, Peiyan Li, Quanyun Zhou, Hangjun Ye, Jiahuan Zhou, and Yadong Mu. Rotvla: Rotational latent action for vision-language-action model. *arXiv preprint arXiv:2605.13403*, 2026c.

Chaoqi Liu, Xiaoshen Han, Jiawei Gao, Yue Zhao, Haonan Chen, and Yilun Du. Oat: Ordered action tokenization. *arXiv preprint arXiv:2602.04215*, 2026a.

Songming Liu, Lingxuan Wu, Bangguo Li, Hengkai Tan, Huayu Chen, Zhengyi Wang, Ke Xu, Hang Su, and Jun Zhu. Rdt-1b: a diffusion foundation model for bimanual manipulation. In *International Conference on Learning Representations*, volume 2025, pp. 29982–30009, 2025.

Yicheng Liu, Zibin Dong, Baijun Ye, Tianyuan Yuan, Tao Jiang, Anqi Yang, Shicheng Cao, Haonan Liu, Yue Sun, Zihan Guo, et al. G0. 5: One autoregressive stream for robot reasoning and action. *arXiv preprint arXiv:2608.11739*, 2026b.

Yiyang Ma, Xingchao Liu, Xiaokang Chen, Wen Liu, Chengyue Wu, Zhiyu Wu, Zizheng Pan, Zhenda Xie, Haowei Zhang, Xingkai Yu, et al. Janusflow: Harmonizing autoregression and rectified flow for unified multimodal understanding and generation. In *2025 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 7739–7751. IEEE, 2025.

Soroush Nasiriany, Sep Nasiriany, Abhiram Maddukuri, and Yuke Zhu. Robocasa365: A large-scale simulation framework for training and benchmarking generalist robots. In *International Conference on Learning Representations*, volume 2026, pp. 98643–98667, 2026.

Xichen Pan, Satya Narayan Shukla, Aashu Singh, Zhuokai Zhao, Shlok Kumar Mishra, Jialiang Wang, Zhiyang Xu, Jiuhai Chen, Kunpeng Li, Felix Juefei-Xu, et al. Transfer between modalities with metaqueries. *arXiv preprint arXiv:2504.06256*, 2025.

Xichen Pan, Aashu Singh, Satya Narayan Shukla, Xiangjun Fan, Shlok Kumar Mishra, and Saining Xie. Repfusion: Leveraging multimodal priors for denoising in representation space. *arXiv preprint arXiv:2606.14700*, 2026.

Karl Pertsch, Kyle Stachowicz, Brian Ichter, Danny Driess, Suraj Nair, Quan Vuong, Oier Mees, Chelsea Finn, and Sergey Levine. Fast: Efficient action tokenization for vision-language-action models. *arXiv preprint arXiv:2501.09747*, 2025.

Weijia Shi, Xiaochuang Han, Chunting Zhou, Weixin Liang, Xi Lin, Luke Zettlemoyer, and Lili Yu. Lmfusion: Adapting pretrained language models for multimodal generation. *Advances in Neural Information Processing Systems*, 38:9163–9182, 2026.

Yating Wang, Haoyi Zhu, Mingyu Liu, Jiange Yang, Hao-Shu Fang, and Tong He. Vq-vla: Improving vision-language-action models via scaling vector-quantized action tokenizers. In *2025 IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 11089–11099. IEEE, 2025.

Wei Wu, Fan Lu, Yunnan Wang, Shuai Yang, Shi Liu, Fangjing Wang, Qian Zhu, He Sun, Yong Wang, Shuailei Ma, et al. A pragmatic vla foundation model. *arXiv preprint arXiv:2601.18692*, 2026.

Lishan Yang, Wenxuan Song, Xi Wang, Pingyue Sheng, Zheng Fang, Ziyang Zhou, Junjie He, Haodong Yan, Jiayi Chen, Nan Sun, et al. 4d-wam: Infusing spatiotemporal awareness into world action models through trajectory fields. *arXiv preprint arXiv:2608.08023*, 2026.

Jinhui Ye, Ning Gao, Senqiao Yang, Jinliang Zheng, Zixuan Wang, Yuxin Chen, Pengguang Chen, Yilun Chen, Shu Liu, and Jiaya Jia. StarVLA-$\alpha$: Reducing complexity in vision-language-action systems. *arXiv preprint arXiv:2604.11757*, 2026.

Tianyuan Yuan, Zibin Dong, Yicheng Liu, and Hang Zhao. Fast-wam: Do world action models need test-time future imagination? *arXiv preprint arXiv:2603.16666*, 2026.

Tony Z Zhao, Vikash Kumar, Sergey Levine, and Chelsea Finn. Learning fine-grained bimanual manipulation with low-cost hardware. *arXiv preprint arXiv:2304.13705*, 2023.

Jinliang Zheng, Jianxiong Li, Zhihao Wang, Dongxiu Liu, Xirui Kang, Yuchun Feng, Yinan Zheng, Jiayin Zou, Yilun Chen, Jia Zeng, et al. X-vla: Soft-prompted transformer as scalable cross-embodiment vision-language-action model. In *International Conference on Learning Representations*, volume 2026, pp. 60580–60606, 2026.

Chunting Zhou, Lili Yu, Arun Babu, Kushal Tirumala, Michihiro Yasunaga, Leonid Shamis, Jacob Kahn, Xuezhe Ma, Luke Zettlemoyer, and Omer Levy. Transfusion: Predict the next token and diffuse images with one multi-modal model. In *International Conference on Learning Representations*, volume 2025, pp. 6446–6469, 2025.

Hongyi Zhou, Weiran Liao, Xi Huang, Yucheng Tang, Fabian Otto, Xiaogang Jia, Xinkai Jiang, Simon Hilber, Ge Li, Qian Wang, et al. Beast: Efficient tokenization of b-splines encoded action sequences for imitation learning. *Advances in Neural Information Processing Systems*, 38: 172934–172959, 2026.

# A Additional Preliminaries

## A.1 Flow-based Action Generation

Following Section 3.1, we train the velocity field with target $\boldsymbol{\epsilon} - \mathbf{A}$, sampling $\tau = 0.999u + 0.001$ with $u \sim \mathrm{Beta}(1.5, 1)$. Action chunks have length $H = 50$, with $D_p = 14$ physical dimensions padded to $D = 32$. Clean padding is zero; Gaussian noise and the flow loss cover all $50 \times 32$ entries, while OAT and execution use only the physical dimensions. Inference integrates from $\tau = 1$ to $\tau = 0$ using

$$\mathbf{A}_{\tau-h} = \mathbf{A}_\tau - h\, v_\Theta(\mathbf{A}_\tau, \tau, o_t). \tag{10}$$

Holding the current velocity fixed gives the one-step endpoint estimate $\widehat{\mathbf{A}}_0 = \mathbf{A}_\tau - \tau v_\Theta(\mathbf{A}_\tau, \tau, o_t)$, used in the analysis in Appendix C.2.

## A.2 Ordered Action Tokenization

OAT (Liu et al., 2026a) appends $M = 8$ learnable registers to the action tokens and encodes them using two transformer blocks of width 256, each with four attention heads. Action tokens attend only to action tokens, while registers attend to all action tokens and preceding registers, including themselves. With queries indexing rows, the attention mask is

$$\mathcal{M}_{\mathrm{OAT}} = \begin{bmatrix} \mathbf{1}_{H\times H} & \mathbf{0}_{H\times M} \\ \mathbf{1}_{M\times H} & \mathrm{tril}(\mathbf{1}_{M\times M}) \end{bmatrix}. \tag{11}$$

The register outputs are normalized and projected to $d_z = 4$ dimensions.

We pretrain a separate OAT for each benchmark by minimizing action reconstruction MSE in the policy's normalized physical action space. Pretraining uses FSQ levels $(8, 5, 5, 5)$ and a four-block decoder of width 256, with batch size 256 and learning rate $5 \times 10^{-5}$. During policy training and inference, JADE uses only the frozen encoder and extracts the bounded, normalized continuous latents before FSQ rounding. The VLM therefore receives $8 \times 4$ continuous features; discrete token IDs and the reconstruction decoder are not used for policy action generation.

# B Implementation Details

## B.1 Action Encoding Interface

OAT encodes only the 14 physical action dimensions into eight four-dimensional latents. The input adapter projects them to width 2048, adds noise-level, register-identity, and action-modality embeddings, and applies layer normalization. VLM registers attend to the observation prefix and all registers, while the prefix does not attend to registers. Prefix keys and values are cached per policy query; register features are recomputed at each denoising step.

## B.2 Semantic Residual Adapter

A shared $2048 \to 64 \to 2048$ SiLU MLP $\Delta_\xi$ adapts the VLM register states:

$$\mathbf{R}_\tau^{(\ell)} = \mathbf{S}_\tau^{(\ell)} + \Delta_\xi(\mathbf{S}_\tau^{(\ell)}). \tag{12}$$

The same adapter is applied to every layer and the final output and is included in $F_\phi$ in Section 3.2. Alignment supervision is applied only at the final output.

## B.3 Fusion Module and Gate

Each expert layer inserts cross-attention after the native attention residual and before the feed-forward network. Action states provide queries; VLM registers provide keys and values through an independent $2048 \to 64 \to 2048$ SiLU mapping $T_\ell$. Cross-attention uses eight heads of width 32 and is normalized separately from native attention. A shared gating MLP uses mean action states, mean fusion updates, and a time embedding to produce a scalar $\tanh$ gate per sample and layer, together with a learned layer-specific bias. The gate output layer and layer-specific biases are zero-initialized, giving zero initial fusion residuals.

### B.4 Joint Optimization

We minimize $\mathcal{L}_{\text{flow}} + 0.1\mathcal{L}_{\text{align}}$. Both losses update the VLM and input and residual adapters; flow matching additionally trains the action expert, fusion modules, and gates. Alignment uses a $2048 \to 4$ linear head with SmoothL1 loss averaged over registers and latent dimensions. OAT remains frozen, with stop-gradient applied to its outputs. The alignment head and clean latent targets are used only during training.

## C Mathematical Analysis of Inference

### C.1 Joint Denoising and Few-Step Integration

Fix an observation $o_t$ and an initial noise sample. Introduce forward sampling time $s = 1 - \tau$ and write $\mathbf{x}(s) = \mathbf{A}_{1-s}$, so that

$$\frac{d\mathbf{x}}{ds} = \mathbf{f}(\mathbf{x}, s), \qquad \mathbf{f}(\mathbf{x}, s) = -v_\Theta(\mathbf{x}, 1 - s, o_t). \tag{13}$$

Assume that $\mathbf{f}$ is $K$-Lipschitz in $\mathbf{x}$ under the Frobenius norm on a region containing the exact and numerical trajectories, and that $\|\mathbf{x}''(s)\|_F \le B$ for $s \in [0, 1]$. The analysis below concerns numerical integration error relative to the exact solution of this learned field.

**Euler discretization bound.** Consider positive Euler steps $h_i$ satisfying $\sum_{i=0}^{N-1} h_i = 1$, and let $h_* = \max_i h_i$. Starting from the same initial state, the endpoint error satisfies

$$\|\mathbf{x}_N - \mathbf{x}(1)\|_F \le \frac{Bh_*}{2}\Gamma(K), \qquad \Gamma(K) = \begin{cases} (e^K - 1)/K, & K > 0, \\ 1, & K = 0. \end{cases} \tag{14}$$

*Proof.* Let $s_i = \sum_{j<i} h_j$ and $e_i = \|\mathbf{x}_i - \mathbf{x}(s_i)\|_F$. Taylor's integral remainder bounds the local truncation error by $Bh_i^2/2$. Lipschitz continuity therefore gives $e_{i+1} \le (1 + Kh_i)e_i + Bh_i^2/2$, with $e_0 = 0$. For $K > 0$, using $h_i^2 \le h_* h_i$ and unrolling yields

$$e_N \le \frac{Bh_*}{2K}\left(\prod_{i=0}^{N-1}(1 + Kh_i) - 1\right) \le \frac{Bh_*}{2K}(e^K - 1),$$

where the last inequality follows from $1 + Kh_i \le e^{Kh_i}$ and $\sum_i h_i = 1$. For $K = 0$, summing the local errors gives $e_N \le (B/2)\sum_i h_i^2 \le Bh_*/2$.

**Implication for fewer steps.** For uniform steps, $h_* = 1/N$ and the bound becomes $B\Gamma(K)/(2N)$. If joint denoising reduces $B\Gamma(K)$ over the trajectories of interest, fewer steps suffice to meet the same bound on discretization error. Here, $B$ bounds the change in velocity along the trajectory, including changes in both magnitude and direction. This provides a conditional explanation of few-step integration; VLM participation alone does not establish a reduction in $B\Gamma(K)$, and integration error is distinct from action-prediction error or task success.

### C.2 Adaptive Sampling Rule and Basic Properties

We analyze the rule in Section 3.4 in exact arithmetic. Assume finite velocity predictions, $0 \le \beta < 1$, $\lambda_{\text{step}} > 0$, $\varepsilon_c > 0$, $0 < h_{\min} \le h_{\max}$, and $0 < h_0 \le 1$. Each policy query starts at $\tau_0 = 1$ with $\mathbf{m}_{-1} = \mathbf{s}_{-1} = \mathbf{0}$. Every evaluation updates the moments using the current correction; the first step uses the fixed size $h_0$, while subsequent steps are selected from the updated indicator.

**Correction and moment statistics.** Let $\mathbf{v}_i = v_\Theta(\mathbf{A}_{\tau_i}, \tau_i, o_t)$ and define the one-step Euler endpoint estimate $\widehat{\mathbf{A}}_{0,i} = \mathbf{A}_{\tau_i} - \tau_i \mathbf{v}_i$. The correction satisfies

$$\mathbf{q}_i = -\tau_i \mathbf{v}_i^{\text{phys}} = \widehat{\mathbf{A}}_{0,i}^{\text{phys}} - \mathbf{A}_{\tau_i}^{\text{phys}}. \tag{15}$$

Thus, $\mathbf{q}_i$ is the displacement toward this estimate in physical action dimensions. Since it includes the remaining time $\tau_i$, it can change even when the predicted velocity is constant.

The first and second moments are

$$\mathbf{m}_i = \beta \mathbf{m}_{i-1} + (1-\beta)\mathbf{q}_i, \qquad \mathbf{s}_i = \beta \mathbf{s}_{i-1} + (1-\beta)\mathbf{q}_i^{\odot 2}.$$

Writing $\mathbf{V}_i = \mathbf{s}_i - \mathbf{m}_i^{\odot 2}$, the indicator is

$$c_i = \|[\mathbf{V}_i]_+\|_F^{1/2} = \left( \sum_{j=1}^{H} \sum_{k=1}^{D_p} [V_{i,jk}]_+^2 \right)^{1/4}. \tag{16}$$

The norm is taken before the square root. Statistics are computed per sample over physical action dimensions, while Euler updates use the full action tensor.

**Nonnegativity and initialization.** The moment difference obeys

$$\mathbf{V}_i = \beta \mathbf{V}_{i-1} + \beta(1-\beta)(\mathbf{q}_i - \mathbf{m}_{i-1})^{\odot 2}. \tag{17}$$

*Proof.* Substituting the two moment updates into $\mathbf{s}_i - \mathbf{m}_i^{\odot 2}$ and collecting terms gives the identity. Since $\mathbf{V}_{-1} = \mathbf{0}$ and both coefficients are nonnegative, induction establishes $\mathbf{V}_i \geq \mathbf{0}$ elementwise. The positive-part operation therefore only guards against floating-point error.

The moments are not bias-corrected. For a constant correction sequence $\mathbf{q}_i = \mathbf{q}$, geometric summation gives $\mathbf{m}_i = (1-\beta^{i+1})\mathbf{q}$ and $\mathbf{s}_i = (1-\beta^{i+1})\mathbf{q}^{\odot 2}$, hence

$$\mathbf{V}_i = \beta^{i+1}(1-\beta^{i+1})\mathbf{q}^{\odot 2}. \tag{18}$$

For $0 < \beta < 1$, even constant nonzero corrections produce a positive early indicator. Thus, $c_i$ reflects both correction variation and the transient effect of zero initialization.

**Step sizes and finite termination.** For each subsequent step with $\tau_i > 0$, define $\bar{h}_i = \mathrm{clip}(\lambda_{\mathrm{step}}/(c_i + \varepsilon_c), h_{\min}, h_{\max})$ and $h_i = \min\{\tau_i, \bar{h}_i\}$. Then

$$0 < h_i \leq \min\{\tau_i, h_{\max}\}, \qquad 0 \leq \tau_{i+1} < \tau_i. \tag{19}$$

The flow time decreases without crossing zero. For fixed $\tau_i$, the step size is nonincreasing in $c_i$, with possible plateaus due to clipping. Each nonterminal adaptive step is at least $h_{\min}$; the terminal step may be smaller.

The number of velocity evaluations satisfies

$$1 + \left\lceil \frac{1-h_0}{h_{\max}} \right\rceil \leq N_{\mathrm{eval}} \leq 1 + \left\lceil \frac{1-h_0}{h_{\min}} \right\rceil. \tag{20}$$

*Proof.* The first evaluation advances by $h_0$, leaving an interval of length $1 - h_0$. Each later step is at most $h_{\max}$, giving the lower bound. Each nonterminal adaptive step advances by at least $h_{\min}$, and the next step exhausts any remaining interval no larger than $h_{\min}$, giving the upper bound. For Balanced, $h_0 = h_{\min} = 0.1$ and $h_{\max} = 0.5$ yield $3 \leq N_{\mathrm{eval}} \leq 10$. Moment updates and step selection require no additional model evaluations.

**Scope of the guarantees.** Equation (14) controls integration error under trajectory regularity assumptions and depends on the actual maximum step size. The adaptive-rule analysis establishes nonnegative moment statistics, monotone time progression, and finite termination. It does not establish $c_i$ as a velocity-field curvature measure or a calibrated local-error estimator, and therefore does not certify an integration-error tolerance. The performance–latency trade-off is evaluated empirically in Section 4.4.

# D Experimental Details

## D.1 Simulation Benchmarks and Metrics

**RoboTwin 2.0.** We train one multi-task policy on 50 clean demonstrations for each of 50 bimanual tasks, giving 2,500 demonstrations. Evaluation uses 100 trials per task under each of the Clean and Random conditions, giving 5,000 trials per condition. Random scenes are not included in training. For each condition we average task success rates; the reported overall average is the arithmetic mean of Clean and Random. Because every task has the same trial count, the task average equals the episode-weighted success rate within that condition.

**RoboDojo.** Training uses $3{,}500$ demonstrations from $35$ training tasks. Evaluation covers $42$ tasks in five categories: Generalization, Precision, Long-Horizon, Memory, and Open. The full evaluation expands task variants across seeds $0, 1, 2$ into $162$ evaluation cells and $6{,}300$ trials. Generalization contributes $1{,}800$ trials, Precision and Long-Horizon $1{,}200$ each, Memory $900$, and Open $1{,}200$. We report the official score on a 0–100 scale and full-task success rate in percent. The overall result averages the five category means with equal weight:

$$\mathrm{Metric}_{5\mathrm{D}} = \frac{1}{5}\sum_{c=1}^{5} \mathrm{Metric}_c \,. \tag{21}$$

This category average differs from an episode-weighted average because category trial counts differ. For example, the archived ten-step JADE run has $825/6300 = 13.10\%$ episode-weighted success, while the category-weighted value reported in the main text is $12.67\%$.

### D.2 Per-Task Results

**RoboTwin 2.0.** Table 4 reports JADE's per-task success rates on all 50 RoboTwin 2.0 tasks under Clean and Random settings, using ten denoising steps and 100 trials per task and setting.

**RoboDojo.** Table 5 reports the Score and success rate of JADE on each of the $42$ RoboDojo tasks with ten denoising steps. Every task includes $150$ trials across seeds $0$, $1$, and $2$. For each Generalization task, the standard and randomized variants contribute $75$ trials each and are averaged with equal weight. Each category mean averages the tasks within that category, and the last row averages the five category means as in Table 2.

### D.3 Preprocessing and Training

Images are resized with padding to $224 \times 224$. Simulation actions comprise 12 joint displacements relative to the current state and two absolute gripper commands. OAT fitting, policy training, and inference share the same quantile normalization statistics. RoboTwin uses a consistent red/blue channel swap during training and inference, while RoboDojo retains RGB order.

Both benchmarks use a 60,000-update cosine learning-rate schedule; RoboTwin training stops after 20,000 updates. Unless otherwise specified, inference uses ten Euler steps to generate a 50-action chunk and executes the first 25 actions before replanning.

### D.4 Adaptive Inference

JADE Balanced uses $h_0 = h_{\min} = 0.1$, $h_{\max} = 0.5$, $\beta = 0.3$, $\varepsilon_c = 10^{-8}$, and $\lambda_{\mathrm{step}} \approx 0.4217$. Moment statistics are reset for each policy query. Sampling uses at most ten model evaluations, while the execution horizon remains fixed at 25. We select $\lambda_{\mathrm{step}}$ to target a mean NFE of five on stored RoboDojo observations, without using success labels, and use the same configuration on both benchmarks. Mean NFE is approximately 4.995 on RoboTwin and 4.985 on RoboDojo. Appendix C.2 provides the mathematical analysis.

## E Computational Cost and Timing

**Training cost.** We compare native $\pi_{0.5}$ and JADE on two NVLink-connected H100 80GB GPUs with BF16, FSDP, global batch size 2, and identical inputs and optimization settings. After five warm-up updates, we time 30 synchronized updates, including forward, backward, optimizer, and EMA computations but excluding compilation and data loading. Median update time increases from 162.7 to 200.6 ms (+23.3%). Matrix/convolution FLOPs per sample increase from 17.349 to 17.448 TF (+0.57%), counting forward, backward, and recomputation passes with two operations per multiply-add. These measurements describe the microbenchmark rather than full-scale training.

**NFE evaluation subsets.** We use ten tasks per benchmark and 50 trials per task and condition, evaluating RoboTwin under both Clean and Random settings. Within each model, fixed-NFE conditions share the same checkpoint, prediction horizon of 50, and execution horizon of 25. The




Figure 6: Experimental setup of the dual-arm robotic platform.

RoboTwin tasks are adjust bottle, move pill bottle to pad, place cans in plastic box, rank blocks by RGB, move can to pot, pick diverse bottles, scan object, rank blocks by size, move stapler to pad, and hang mug. The RoboDojo tasks are put bottles into dustbin, stack bowls, fold clothes, pour balls into vase, classify objects, make Kong, build tower, match and pick from conveyor, insert tubes, and general pickup. These subsets differ from those in Appendix F.

**Inference latency.** Model-side latency per policy query excludes communication, simulation, and robot execution. Generating a 50-action chunk takes approximately 74 ms for ten-step native $\pi_{0.5}$, compared with 203, 132, and 78 ms for ten-step JADE, JADE Balanced (about five evaluations), and two-step JADE, respectively.

## F Step-wise Semantic Update Study

The study in Section 4.5 uses ten RoboDojo tasks, two from each capability category. The Generalization tasks are stack bowls and pack objects into box, the Precision tasks are insert tubes and insert key, the Long-Horizon tasks are classify objects and fill egg holder, the Memory tasks are cover blocks and imitate sorting sequence, and the Open tasks are general pickup and classify objects by language.

All five variants use the same trained checkpoint and change only how the VLM features are computed at inference. Dynamic recomputes the features from the current action at every step, and Lagged reuses those computed at the previous step. Fixed recomputes them from the initial noise with the current flow time, and Static computes them once from the initial noise. RF-Off sets the fusion residuals to zero, so the action expert runs without VLM features.

## G Real-World Experiments

**Platform and training.** We use a PiPER dual-arm robot equipped with one overhead camera and two wrist cameras. The policy receives RGB images, robot states, and a language instruction, and predicts chunks of 50 actions. Each action contains 12 absolute joint-angle targets and two gripper commands. For each task, native $\pi_{0.5}$ and JADE are fine-tuned on the same 100 teleoperated demonstrations. JADE additionally uses a task-specific OAT encoder, which is pretrained on the demonstrations and frozen during policy training.

**Tasks.** We evaluate four tasks: **Tool Placement**, in which the robot picks up a screwdriver and places it in a tool case; **Bimanual Handover**, in which one arm picks up a banana and transfers it to

Table 4: Per-task success rates (%) of JADE on RoboTwin 2.0 with ten denoising steps. Each entry is computed over 100 trials. The last row averages all 50 tasks.

| Task | Clean | Random | Task | Clean | Random |
|---|---|---|---|---|---|
| adjust bottle | 100 | 89 | place can basket | 80 | 30 |
| beat block hammer | 90 | 34 | place cans plasticbox | 86 | 82 |
| blocks ranking rgb | 85 | 53 | place container plate | 95 | 91 |
| blocks ranking size | 43 | 24 | place dual shoes | 72 | 53 |
| click alarmclock | 88 | 76 | place empty cup | 93 | 91 |
| click bell | 95 | 90 | place fan | 73 | 40 |
| dump bin bigbin | 93 | 76 | place mouse pad | 43 | 23 |
| grab roller | 100 | 97 | place object basket | 75 | 40 |
| handover block | 62 | 6 | place object scale | 85 | 59 |
| handover mic | 99 | 15 | place object stand | 95 | 66 |
| hanging mug | 17 | 19 | place phone stand | 73 | 47 |
| lift pot | 93 | 30 | place shoe | 87 | 68 |
| move can pot | 83 | 24 | press stapler | 84 | 70 |
| move pillbottle pad | 90 | 55 | put bottles dustbin | 69 | 54 |
| move playingcard away | 91 | 82 | put object cabinet | 43 | 28 |
| move stapler pad | 35 | 19 | rotate qrcode | 84 | 25 |
| open laptop | 96 | 77 | scan object | 57 | 48 |
| open microwave | 91 | 42 | shake bottle | 100 | 96 |
| pick diverse bottles | 68 | 42 | shake bottle horizontally | 100 | 99 |
| pick dual bottles | 90 | 56 | stack blocks three | 75 | 24 |
| place a2b left | 85 | 53 | stack blocks two | 95 | 74 |
| place a2b right | 82 | 50 | stack bowls three | 78 | 56 |
| place bread basket | 81 | 67 | stack bowls two | 93 | 77 |
| place bread skillet | 79 | 66 | stamp seal | 54 | 38 |
| place burger fries | 96 | 90 | turn switch | 45 | 11 |
| | | | Average | 79.3 | 54.4 |

the other arm for placement on a plate; **Object Arrangement**, in which the robot places a blue bowl on a plate, a green bowl inside the blue bowl, and a red block inside the green bowl; and **Tabletop Organization**, in which the robot moves a bottle, a bowl, a red object, and a white block into a green tray.

**Evaluation.** Each policy is evaluated over 30 trials per task. We report the success rate for each task and the unweighted mean across the four tasks.

Table 5: Per-task Score and success rate (SR, %) of JADE on RoboDojo with ten denoising steps. Each task includes 150 trials. The last row averages the five category means.

| Task | Score | SR | Task | Score | SR |
|---|---|---|---|---|---|
| *Generalization* | | | *Long-Horizon* | | |
| stack bowls | 54.60 | 48.00 | put bottles into dustbin | 96.70 | 94.67 |
| push T | 0.00 | 0.00 | fill pen holder | 41.90 | 16.67 |
| pack objects into box | 29.93 | 7.33 | classify objects | 43.63 | 33.33 |
| fold clothes | 59.07 | 54.67 | play tic tac toe | 15.10 | 2.00 |
| hang mugs | 9.40 | 0.67 | fill egg holder | 6.50 | 0.00 |
| sweep blocks | 3.33 | 3.33 | organize table | 45.67 | 2.67 |
| pour liquid into cup | 23.33 | 23.33 | make Kong | 42.67 | 42.67 |
| make toast | 16.33 | 6.00 | play stacking toy | 0.20 | 0.00 |
| arrange largest number | 5.00 | 1.33 | *Category mean* | *36.55* | *24.00* |
| sort nesting dolls by size | 7.33 | 7.33 | *Memory* | | |
| store laptop and headphones | 22.53 | 6.00 | cover blocks | 20.63 | 14.67 |
| stack blocks | 16.27 | 10.67 | match and pick from conveyor | 22.67 | 22.67 |
| *Category mean* | *20.59* | *14.06* | swap blocks | 0.00 | 0.00 |
| *Precision* | | | swap T | 0.00 | 0.00 |
| fasten screws | 22.93 | 8.00 | press by number | 0.67 | 0.67 |
| plug in charger | 0.00 | 0.00 | imitate sorting sequence | 3.23 | 1.33 |
| insert tubes | 35.60 | 16.67 | *Category mean* | *7.87* | *6.56* |
| pour balls into vase | 44.67 | 44.67 | *Open* | | |
| play xylophone | 1.33 | 1.33 | align blocks | 0.00 | 0.00 |
| deposit coin | 6.80 | 3.33 | general pickup | 18.67 | 18.67 |
| insert key | 13.80 | 0.00 | stack blocks by language | 2.53 | 0.00 |
| build tower | 62.13 | 56.67 | solve equation | 0.00 | 0.00 |
| *Category mean* | *23.41* | *16.33* | classify objects by language | 2.87 | 0.67 |
| | | | pick from conveyor by image | 0.00 | 0.00 |
| | | | store tools in toolbox | 0.33 | 0.00 |
| | | | pour by language | 0.00 | 0.00 |
| | | | *Category mean* | *3.05* | *2.42* |
| | | | Five-category mean | 18.29 | 12.67 |

Table 6: Training configuration for JADE on the simulation benchmarks.

| Setting | RoboTwin | RoboDojo |
|---|---|---|
| Training demonstrations | 2,500 | 3,500 |
| Optimizer updates | 20,000 | 60,000 |
| Global batch size | 256 | 256 |
| Hardware | 8× A800 80GB | |
| Precision / sharding | BF16 / FSDP | |
| Optimizer | AdamW, $(\beta_1, \beta_2) = (0.9, 0.95)$ | |
| Peak / minimum learning rate | $2.5 \times 10^{-5}$ / $2.5 \times 10^{-6}$ | |
| Warm-up updates | 1,000 | |
| Alignment weight $\lambda_{\text{align}}$ | 0.1 | |